Discrete Comput Geom 11:163–191 (1994)

Discrete & Computational
Geometry


# Approximating the Minimum Weight Steiner Triangulation

David Eppstein

Department of Information and Computer Science,
University of California, Irvine, CA 92717, USA
eppstein@ics.uci.edu

**Abstract.** We show that the length of the minimum weight Steiner triangulation (MWST) of a point set can be approximated within a constant factor by a triangulation algorithm based on quadtrees. In $O(n \log n)$ time we can compute a triangulation with $O(n)$ new points, and no obtuse triangles, that approximates the MWST. We can also approximate the MWST with triangulations having no sharp angles. We generalize some of our results to higher-dimensional triangulation problems. No previous polynomial-time triangulation algorithm was known to approximate the MWST within a factor better than $O(\log n)$.

## 1. Introduction

Optimal triangulation has furnished a number of problems of longstanding interest in computational geometry. These problems have applications to cartography, spatial data analysis, and finite-element methods. Optimization criteria for which efficient algorithms are known include maximizing the minimum angle [20], [24], minimizing the maximum angle [6], minimizing the minimum angle [7], minimizing the maximum aspect ratio [3], and minimizing the maximum edge length [5].

The most longstanding open problem in computational geometry is the complexity of another optimal triangulation problem, the *minimum weight triangulation* (MWT), in which the optimization criterion is the sum of the edge lengths. Indeed, this seems to have been known as the "optimal triangulation" for some time. The MWT problem is included in Garey and Johnson's famous list of problems neither known to be NP-complete nor known to be solvable in polynomial time [9]. It is known that many triangulation algorithms will not correctly solve the MWT problem [17]. We do not resolve the status of this question.

The algorithms and problems cited above search for triangulations in which the vertex set of the triangulation is exactly the set of input points. Many of these problems can be extended to *Steiner* triangulation problems, in which the vertex set must be a superset of the input points. The additional vertices are known as Steiner points. For the application of triangulation problems to finite-element mesh generation, Steiner triangulation is more natural than non-Steiner triangulation because the problem is simply to divide space into a number of (triangular) cells, and extra cell boundary points are not a problem. However, it is important that the number of additional Steiner points be relatively small, as this number directly affects the time to solve the resulting finite-element system.

No Steiner triangulation problem is known to be exactly solvable in polynomial time, but a number of approximations have been published. In particular, a Steiner triangulation in which all angles are bounded between 36° and 80° can be found, using an algorithm based on *quadtrees* [1]. The number of Steiner points may vary with the geometry of the input, but the algorithm uses within a constant factor of the optimal number of points, and takes time polynomial in the total output size. Similar methods can be used to find a triangulation with $O(n)$ Steiner points in which no angle is obtuse [1]. We revisit these two triangulations later.

For the minimum weight triangulation problem, it is not immediately clear that adding Steiner points can reduce the total edge length of the MWT. However, in fact we can show that the minimum weight Steiner triangulation (MWST) of $n$ points can have a total weight $\Omega(n)$ times smaller than the MWT. Algorithms are known for approximating the MWT and the MWST [2], [16], [21], [25], but the best such algorithms have a total length that is $O(\log n)$ times the length of the MWT or MWST. In this paper we give the first constant-factor approximation algorithms for the MWST.

### *1.1. New Results*

We describe algorithms for several triangulations, all based on the quadtree recursive space-partitioning data structure. Each of these triangulations has a total length that is $O(1)$ times the length of the MWST. Our algorithms can be implemented to take time $O(n \log n + k)$, where $k$ is the number of Steiner points.

- We describe a simple Steiner triangulation based on quadtrees, in which each quadtree square is split until no square contains more than one input point, along with a balancing condition on the sizes of neighboring squares. This triangulation can use a nonpolynomial number of Steiner points, but is easy to implement and should perform well in practice. We prove that the total length of this triangulation is $O(1)$ times the MWST length.
- We modify the above algorithm to stop dividing squares below $O(\log n)$ levels in the quadtree. This reduces the number of Steiner points to $O(n \log n)$, while preserving the approximation to the MWST.

- We show that a variation of the algorithm from [1], in which some quadtree levels are "shortcut" to reduce the number of Steiner points to $O(n)$, and in which no triangle contains an obtuse angle, also approximates the MWST.
- We modify a triangulation from [1] which avoids sharp angles, and uses a number of Steiner points within a constant factor of the optimum needed to avoid sharp angles. The modified triangulation retains these properties and approximates the MWST. As in [1], a nonpolynomial number of Steiner points may be needed.
- We approximate the MWST of a convex polygon (not just point set) using $O(n)$ Steiner points.

In addition, we use our quadtree characterization of the MWST weight in order to prove some other results on minimum triangulation:

- We prove that the MWST can have a total weight that is $\Theta(\log n)$ times the weight of the minimum spanning tree (which we abbreviate MST; similarly the minimum weight Steiner tree would be the MSST). The proofs of this and the following MWST properties are based on our result that quadtrees approximate the MWST.
- We prove that the MWT can have a total weight that is $\Theta(n)$ times the weight of the MWST. Even for convex polygons or point sets in convex position, the MWT can have a total weight $\Theta(\log n)$ times the MWST weight.
- We prove that, for convex polygons, the MWST can be approximated by a Steiner triangulation in which all Steiner points are on the polygon boundary.

Since minimum length triangulation has been motivated in part by finite-element mesh generation, it is noteworthy that we can find triangulations which are good by other measures of their applicability to finite-element analysis, and which also approximate the MWST.

The relation between MWT and MWST weight shows the importance of allowing Steiner points in a triangulation. The relation between MWST and MST weight is relevant because previous Steiner triangulation algorithms proved their performance by comparing the triangulation length to the MST; since any triangulation spans all vertices, and since the MST approximates the MSST [4], it follows that the MWST weight is at least a constant factor times that of the MST. However, the result above shows that such a proof can never lead to an approximation factor better than $O(\log n)$. Our proof of this result is algorithmic: we find a point set for which our algorithms produce a triangulation with this length. The result then follows from the correctness of our approximation to the MWST.

The result on convex polygons is interesting because, although it is not known how to find the MWST in polynomial time, the optimal triangulation using only boundary Steiner points can be found easily using a dynamic programming technique. We conjecture that, for convex polygons, the MWST in fact only needs boundary Steiner points.

### 1.2. *Related Work*

Quadtrees have been in use for well over a decade, with a number of applications [23] including finite-element mesh generation [18], [22], [26]. The first theoretical analysis of quadtree triangulations appears in the recent work of Bern *et al.* [1] in which it is proved that such triangulations can be used to avoid sharp and obtuse angles, with a number of points within a constant factor of optimal. These restrictions on the angles in the triangulation are motivated by considerations from finite-element analysis. We modify these algorithms slightly—the main difference is in a choice of several initial squares covering the points rather than a single large square—and show that the modified triangulations approximate the MWST. The unmodified algorithms turn out to be within an $O(\log n)$ factor of the MWST weight.

Previous attempts at approximating the MWT and MWST have followed two approaches. In the first approach, various workers have attempted to show that certain other standard triangulations approximate the MWT. For instance, it was at one point believed that the Delaunay triangulation actually achieved the minimum weight; however, it has since been shown that it can be as far as $\Omega(n)$ from the optimum [11], [19]. This is pessimal since any triangulation achieves $O(n)$ times the minimum weight [11]. Similarly, the greedy triangulation [10], [15] has been proposed as an approximation to the MWT; however, the approximation factor can be as bad as $\Omega(\sqrt{n})$ [13], [19]. On the other hand, for convex polygons the greedy triangulation offers an easily computed approximation to the MWT [14], [15].

The second approach to approximate MWT problems uses the insight that polygon MWT is significantly easier than the point-set MWT or MWST problems. The exact MWT of a simple polygon can be found by dynamic programming in time $O(n^3)$ [10], [12]. If the polygon is convex, a triangulation of weight $O(\log n)$ times the polygon's perimeter can be found by the *ring heuristic* of repeatedly connecting all pairs of adjacent even-numbered vertices [21], and as mentioned above a constant-factor approximation to the MWT can be computed in linear time [14], [15].

Lingas [16] suggested starting with polygonal regions formed by combining the convex hull with the MST of a point set, and then computing the optimal triangulations of these regions. He showed that this leads to a triangulation with a total length of $O(|MWT| \log n + nx \log n)$, where $x$ is the length of the longest edge in the triangulation. However, this formula does not show an approximation ratio better than $O(n)$.

Plaisted and Hong [21] used a more complicated method to partition the points into convex polygons. Then the ring heuristic can be used to triangulate the polygons, achieving a total triangulation length of $O(\log n)$ times the MWT length. The Plaisted–Hong algorithm has recently been implemented with a running time of $O(n^2 \log n)$ [25]. Since they do not use Steiner points, the Plaisted–Hong triangulation can be as far as $\Omega(n)$ from the MWST weight.

Clarkson [2] generalizes the ring heuristic to nonconvex polygons, by allowing the addition of Steiner points. His method, together with Lingas' partition into

polygons, achieves an $O(\log n)$ approximation to the MWST; until the present work this was the best such approximation known.

### *1.3. Organization of this Paper*

In the next section we describe the simplest version of our quadtree algorithm, and prove that it approximates the MWST. The algorithm may have nonpolynomial output size (number of Steiner points), but we show that the running time is polynomial in the input and output sizes. The proof of approximation first relates the triangulation to the MWT, and then shows that adding further Steiner points would only increase the length of the triangulation produced by our algorithm; therefore it also approximates the MWST.

In the third section we describe a number of modifications of our algorithm to produce triangles satisfying certain other properties. We extend the algorithms of [1], in which all angles are within certain bounds, to new triangulations that both approximate the MWST and satisfy the angle bounds. We also show that $O(n)$ Steiner points suffice to approximate the MWST, producing the first-known polynomial-time approximation to the MWST.

In the fourth section we prove a number of results about MWSTs by analysing the behavior of the quadtree algorithm on various point sets. In the fifth section we introduce higher-dimensional extensions of the MWT and MWST problems, discuss some difficulties in extending our results, and prove an $O(\log n)$-approximation to the minimum edge length Steiner triangulation in any dimension.

## 2. Basic Quadtree Triangulation

### *2.1. Triangulation Algorithm*

A *quadtree* is a recursive partition of a region of the plane into axis-aligned squares. One square, the *root*, covers the region that is to be partitioned. Each square may be divided into four *child* squares, by splitting it with horizontal and vertical line segments through the center of the square. Each child has a size (length of the sides of the square) proportional to half the parent's size. Thus the collection of squares forms a tree, with smaller squares at lower levels of the tree. A *leaf square* is one that has not been further subdivided into children.

Each leaf square in the quadtree has a set of *neighbors*, those leaf squares sharing either corner vertices or portions of sides with the square. A neighbor is *orthogonally adjacent* if it shares a portion of the square's sides, and *diagonally adjacent* if it only shares a corner point.

As in [1], we maintain an additional *balance condition* in the quadtrees we construct: the orthogonally adjacent neighbors of any leaf square must be at most twice the size of the square, and at least half the size of the square. Equivalently, each line segment forming the boundary of a leaf square can be divided into at most two parts by subdivision of neighboring squares. We say a quadtree is

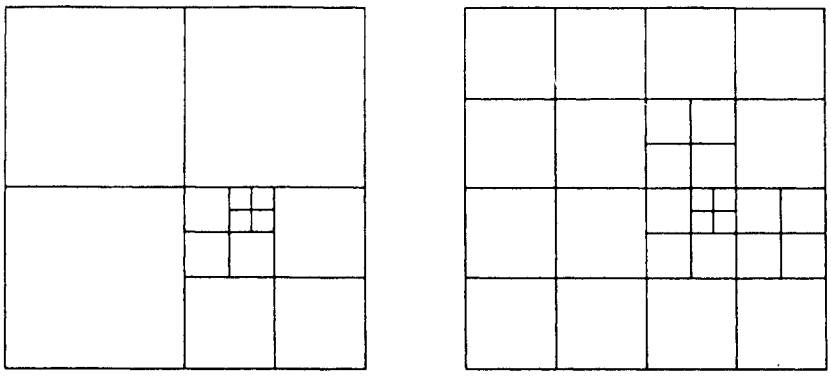



**Fig. 1.** Balance condition: (a) unbalanced quadtree; (b) additional subdivisions restore balance.

*balanced* if the balance condition is true. Unbalanced and balanced quadtrees are shown in Fig. 1(a) and (b).

Our triangulation algorithm first builds a quadtree covering the set of input points. Then it triangulates a rectangular region of the quadtree, including all line segments bounding leaf squares as edges in the triangulation. These edges are augmented by diagonal edges in squares not containing input points, and edges from the input points to the quadtree vertices surrounding them in the squares containing them.

We first find any root square containing the entire point set. We then recursively subdivide squares of the quadtree, maintaining the balance condition. We subdivide any square containing two input points in its interior or on its boundary. Whenever two neighboring squares violate the balance condition, we must also split the larger of the two squares, regardless of whether it contains an input point. At the end of this process, each input point is alone in its square. Note that this phase of the algorithm is inherently nonpolynomial, because two close together points may require a number of subdivisions unrelated to the total number of points.

We next cut the quadtree by adding the boundary of the convex hull of the input point set. We add a Steiner vertex at each point where the boundary crosses a line segment of the quadtree. Any quadtree squares entirely outside this boundary are removed, as are the exterior portions of squares crossed by the boundary.

Finally, we include as Steiner points all corners of quadtree squares, and triangulate the resulting point set. Our triangulation includes as edges the sides of each square. Each leaf square has $O(1)$ vertices on its boundary, because of the balance condition and because the convex hull of the input can only cross a square at most four times. Each square may also contain an input point in its interior or on its boundary. We simply triangulate each square with the minimum length triangulation of those $O(1)$ points, in constant time per square.

Note that the order in which subdivisions are performed does not change the final quadtree. Therefore adding additional points to the input set (within the original convex hull) can only increase the total amount of subdivision and therefore the total length of the triangulation.

### 2.2. *Implementation Details*

A number of details are required to perform the above triangulation algorithm efficiently; these are essentially the same as those required in [1].

In the course of the algorithm we maintain a partially split quadtree. For each square we maintain a data structure with pointers to its parent, its children, and its same-size neighbors. Whenever we split a square, we update the parent and child information. If the new children have neighbors of the same size, the neighbors are either siblings, or they are children of the neighbors of the split square. Therefore we can also maintain the neighbor points in constant time per subdivision.

To maintain the balance condition, we keep a queue of unbalanced neighbor pairs. Whenever we split a square, we determine if any of its parent's unsubdivided neighbors is a neighbor of the split square; if so the neighbor and some child of the split square form an unbalanced pair. This check can be performed in constant time, and it is not hard to see that all unbalanced pairs will be detected in this way. Whenever this queue is nonempty, we pull the top pair off the queue and check if the two squares still violate the balance condition; if so we subdivide the larger square and update the queue as necessary. By emptying the queue before continuing with the algorithm, we perform subdivisions to restore the balance condition before we allow any subdivisions for other reasons.

There are two ways to perform the checks that each input point is alone in its square. The simpler of the two is to add the points one at a time to the quadtree. Each leaf square contains a pointer to the point it contains, or is noted as containing no point. When we add a new point, we trace through the quadtree to find the appropriate leaf, and test if the leaf is empty. If not, we have a pair of points in the same square, and we subdivide further until the points are separated. This method works for the truncated quadtree triangulation described later, but it can be inefficient for other quadtree algorithms (including the basic quadtree triangulation described above) because we may have many points at a low level in the tree, and each point takes time proportional to its level.

The second method for managing the input points is to deal with them all at once. Initially, the root square contains a list of all the points. When a box is split, we must divide that list into four parts, one for each of the child boxes. The difficulty here is performing that division efficiently when the points are unevenly divided among the children. We solve this difficulty by maintaining two doubly linked lists, of the points sorted by horizontal and vertical coordinates. We split the square horizontally by moving in simultaneously from both ends of the horizontally sorted list, until we find the division corresponding to the center line of the square. We determine which of the two sides contains the smaller set of

points, and we extract the points in that set one at a time from the vertically sorted list.

This takes time proportional to the size of the smaller set. however, it leaves the vertical list of the smaller set unsorted. Along with the two lists, we maintain, for each point, two integers representing its order in the two lists. Whenever we extract a smaller subset of points from a larger set, we sort the smaller subset using the previously computed integers, then we recompute the integers in the smaller subset to be exactly the ranks in the sorted lists. However, we do not recompute the integers for the points remaining in the larger subset until the size of the subset drops below half of what its size was the last time we performed such a recomputation. In this way the ranking integers can be maintained at a total cost of $O(n \log n)$ over the span of the algorithm. We use these ranking integers to sort the vertical set. If there are $x$ points in the large set, the points will have ranks between 1 and $2x$. If we wish to extract $k$ points, then the sorting can be performed in time $O(k \log(x/k))$ using a combination of bucket sorting (with $k$ buckets) and comparison sorting. The total cost of splitting squares horizontally then also becomes $O(n \log n)$. Vertical splitting is performed in a similar way.

**Theorem 1.** *A quadtree triangulation on $n$ input points, with $k$ Steiner points, can be constructed in time $O(n \log n + k)$.*

*Proof.* The number of Steiner points is proportional to the number of times quadtree squares are subdivided. As described above, the total time to split a square is $O(1)$, together with the cost of splitting the sets of points in the square which can be amortized to a total of $O(n \log n)$. So if the total number of squares produced is $k$, the total time will be $O(n \log n + k)$. □

### 2.3. *Approximation to MWT*

We first show that the quadtree triangulation approximates the minimum weight (non-Steiner) triangulation of the input points. This is strange, because we are approximating a non-Steiner triangulation by a Steiner triangulation. However, as we shall see this step is needed in our proof that the quadtree triangulation approximates the MWST.

We slightly modify the definition of the triangulation we are using: we assume that each input point is contained in a quadtree square with very small side length, say $\varepsilon/n$ for some $\varepsilon$. This is without loss of generality, since the extra splitting needed to ensure this only increases the total edge length of the triangulated quadtree. With this assumption, the boxes containing points have total edge length $\varepsilon$ and can be neglected in the analysis—all the edge length arises from empty boxes. This reduces the number of cases we need to consider.

We now show that the triangulation length can be measured just by the edges in the quadtree, without having to worry in detail about the diagonals added to produce a triangulation.

**Lemma 1.** *Within the portion of the quadtree within the convex hull of the input, the ratio in edge lengths between the triangulated and untriangulated versions of the quadtree is* $O(1)$.

*Proof.* If we sum the perimeters of all unsubdivided squares in the quadtree, each interior edge is counted twice and each exterior edge is counted once. So if each edge is responsible for a total length proportional to twice its side length, all lengths will be covered except for half the exterior edges. (Alternately, imagine each square being responsible for its bottom right edges, with the top left edges taken care of by the square's neighbors except for the exterior edges for which no such neighbor exists).

Within a given unsubdivided square $s$ of side length $l$, we must add diagonals to form a triangulation. There are six possible cases, depending on the number of subdivision points appearing on the sides of $s$ (there are two different cases for squares with two subdivisions). In each of these cases, we charge some of the length in the added diagonals to $s$ itself, and some of the length to the half-size squares causing the subdivision points.

Note that any set of four siblings in the quadtree square can be charged for subdivisions on at most three larger squares. There cannot be four larger squares, surrounding a set of four equal-sized siblings, because the parent of the siblings could only be subdivided because of a pair of points within it, and since we are assuming input points are in negligible-size squares one of the four siblings would have to be small. If there are three larger squares, all four siblings must be present to be charged, and if there are two larger squares they must charge at least three siblings. So if a square can be charged by a factor of $x$ times its length, a square with subdivisions will be able to charge a factor of $2x/3$ times its length per subdivision.

We apply this charging process with $x = 3/4(1 + \sqrt{5} - \sqrt{4})$. Then the total charge for squares with no subdivisions, or with two opposite subdivisions, including the cost for the diagonals and the $2l$ charge for the square itself, is $(11 + \sqrt{2} + 3\sqrt{5})l/4 \approx 4.78l$. The charge for a square with one subdivision is $(9 + 5\sqrt{5} - \sqrt{2})l/4 \approx 4.69l$. The charge for a square with two adjacent subdivisions is $(7 + 3\sqrt{2} + 3\sqrt{5})l/4 \approx 4.49l$. The charge for a square with three subdivisions is $(9 + 7\sqrt{2} - \sqrt{5})l/4 \approx 4.16l$. The charge for a square with four subdivisions is $(7 + 11\sqrt{2} - 5\sqrt{5})l/4 \approx 2.844l$. The desired ratio is obtained by dividing the maximum of these numbers by the $2l$ charge per square of the untriangulated quadtree, obtaining a value of approximately 2.39. □

**Lemma 2.** *The total edge length of the quadtree triangulation is* $O(m)$, *where* $m$ *is the total edge length of the MWT.*

*Proof.* As noted above, we can allocate the edge length of the triangulated quadtree to its individual unsubdivided squares. Each square of side length $l$ is charged with length $4.78l$. By the assumption that squares containing input points

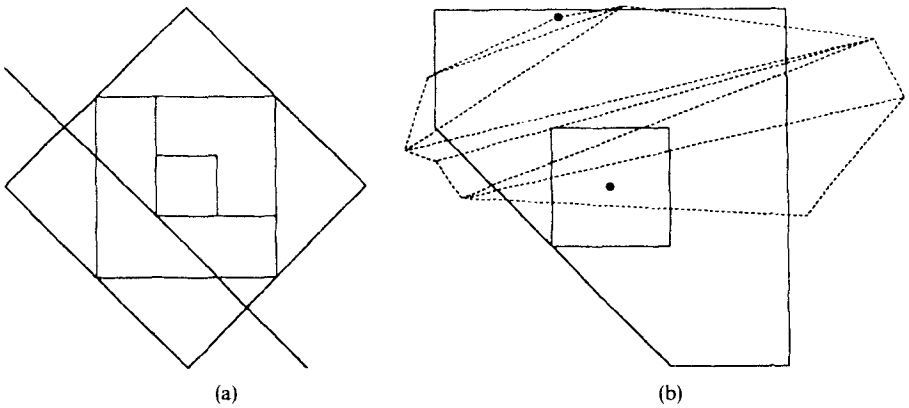



**Fig. 2.** The first to fourth cases of Lemma 2: (a) regions $s$, parent of $s$, $3s$, $3\sqrt{2}d$, and $D$; (b) stepping from triangle to triangle until we find a point in $3s \cap D$.

are small, we only need to worry about the empty squares. We now partition the unsubdivided empty squares into several types. We treat each type in a separate case.

In the following description, $s$ refers to an unsubdivided square of side length $l$ which we are attempting to charge to the MWT, $c$ refers to the center point of $s$, $d$ refers to a square with the same side length and centerpoint, but at an angle of 45° to $s$, and $kd$ or $ks$, for some constant $k$, refers to the concentric squares with side length $kl$. $H$ refers to the convex hull of the input, which is the region covered by the MWT triangles. $D$ refers to a half-plane, with a boundary line at 45° to the sides of $s$, touching $s$ at one corner and entirely containing the quadtree parent of $s$. Some of these regions are depicted in Fig. 2(a).

In the first case, $c$ is in $H$ and the MWT triangle covering $c$ has a vertex in the region $3s \cap D$. We then charge the edge length of $s$ to the two adjacent triangle edges, proportional to their lengths. Any triangle covering $c$ with vertices outside $s$ has an edge of length at least $l$, so by the triangle inequality the two charged edges together have a total length of at least $l$. Each MWT edge is charged at most 12 times for each size of square, since each endpoint can be charged by six squares neighboring the one containing the endpoint. The charge per square $s$ is at most $4.78l$. Each endpoint of an edge can be charged by at most six squares of any given size.

In the second case, $c$ is in $H$ and some orthogonally adjacent sibling $s'$ of $s$ contains an input point $x$ that has side length $s$ and that shares a side with $s$. Note that $s'$ is entirely contained in $3s \cap D$. If $s$ was not covered by the first case, no vertex of the MWT triangle containing $c$ is in $3s \cap D$. Since this triangle is empty, $x$ is outside it. Therefore some edge of the triangle cuts entirely across region $3s \cap D$ and separates $c$ from $x$. We next examine the triangle on the other side of this edge. Again, this triangle either has a vertex in $3s \cap D$ or another edge separates $c$ from $x$ in $3s \cap D$. If we repeat this process, we find a triangle with one vertex $v$ in $3s \cap D$, and with the opposite edge cutting entirely across $3s \cap D$

separating $c$ from $x$. We charge the two triangle edges adjacent to $v$ proportionally to their lengths.The opposite triangle edge must have length at least $l$, so by the triangle inequality this is also true for the sum of the charged edge lengths. Figure 2(b) shows a sequence of triangles ending at one with an endpoint in $3s \cap D$, as described in this case. The charge per square, and the squares that can charge a given edge, are the same as in the first case.

In the third case, $c$ is in $H$ and the MWT triangle covering $c$ has a vertex in the region $3\sqrt{2}d \cap D$. (This region is depicted with $3s$ in Fig. 2(a).) This case is similar to the first case. Because the vertices of the triangle avoid $3s \cap D$, we can charge a pair of edges with total length at least $2l$, so the charge per square relative to the edge's length is smaller by a factor of two. Each endpoint of an edge may be charged by 11 squares of a given size, but six of these potential charging squares were accounted for in the first two cases, leaving five new charges.

In the fourth case, $c$ is in $H$ and $s$ has a diagonally adjacent sibling that contains an input point. This is similar to the second case, in that we step from triangle to triangle to produce a sequence of edges separating $c$ from this input point in $3\sqrt{2}d \cap D$. We then charge a pair of edges with total length at least the length of the last edge in the sequence. The shortest line segment crossing $3\sqrt{2}d \cap D$ and separating $c$ from some point of its diagonal neighbor has length $3\sqrt{2}/2l > 2l$. The charge per square, and the squares that can charge a given edge, are the same as in the third case.

The fifth case covers all remaining squares $s$ that are entirely contained in $H$. Let $s'$ denote the quadtree parent square of $s$, let $c'$ denote the centerpoint of $s'$, and let $d'$ denote the square with side length $2l$ centered on $c'$ at an angle 45° to $s'$. Since $s$ is in $H$, and $c'$ is in $s$, $c'$ is in $H$. If $s$ is not covered by the previous four cases, $s'$ must be empty, and must have been split by the balance condition. Therefore there must be some input point $x$ within the region $3s' \cap 2\sqrt{2}d'$. We first consider the MWT triangle containing $c'$. If it has an endpoint in $3s'$, we charge the edge length of $s$ to a pair of edges with total length at least $2l$, adjacent to that endpoint. Otherwise, some edge of the triangle crosses square $3s$, and separates $c'$ from $x$; this edge must have length at least $2l$. As in the second and fourth cases, we can step from triangle to triangle until we find a pair of edges, with total length at least $2l$, sharing an endpoint within $3s'$. Each endpoint of an edge may be charged by 32 squares of any given size; however, eight of these could also have charged the edge in the first four cases, leaving 24 new charges.

Figures 3(a) and (b) depict the neighborhood of an edge endpoint, depicting the various squares of a given size that can charge it in various cases. Note that if any of the eight nearest-neighboring squares to the point charge it, then certain squares of half the size cannot do so. In this sort of conflict, a larger charge will always be accumulated when these eight squares are subdivided and their children charge the edge, than when they are unsubdivided and charge the edge directly. Therefore the total charge to an edge will be maximized when, for each possible size of square, the 27 outer squares charge the edge and the eight inner ones do not. The total charge per unit edge length will then be $54 \cdot 4.78 < 258.2$: we multiply 27 by the charge per square, divide by two since an edge or pair of edges

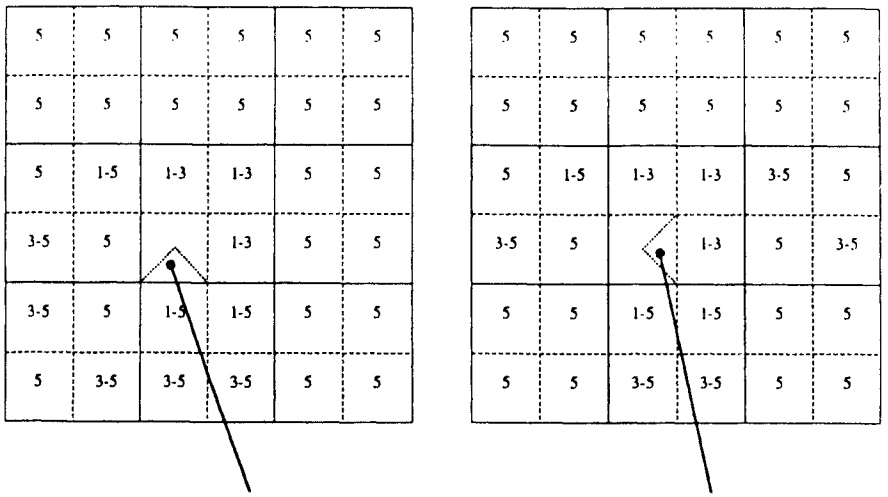



**Fig. 3.** Squares that can charge an edge in various cases of Lemma 2: (a) the endpoint is near the boundary of the parent; (b) the endpoint is near the center of the parent.

can only be charged by squares of half the side length, multiply by two since each edge has two endpoints, and multiply by two again to sum the geometric series formed by considering all possible sizes of squares.

In the remaining cases, square $s$ is crossed by the boundary of the convex hull of the input (if $s$ were entirely outside the convex hull, it would have been removed from the quadtree before triangulation and so would not contribute to the total edge length). Square $s$ can be crossed at most four times by such line segments, and each one cuts off at least one corner of $s$, so there can be at most 12 points on the boundary of $s$, and $s$ can be triangulated with total length at most $(2 + 4\sqrt{2})l$ simply by adding a point in the center.

In the sixth case we consider squares in which the convex hull boundary crosses $s$ in a line segment such that no corner of $s$ is within distance $l/3$ of both line segment endpoints. We charge the edge length in $s$ to that segment, which must have length at least $l/3$. Each unit of convex hull boundary is then charged $(2 + 4\sqrt{2})3 < 23$ times.

In the seventh case, $s$ is crossed near a corner but the center of $s$ is outside the convex hull of the input. Then $s$ together with the crossing segment already forms a triangle, and we charge the portion of convex hull boundary crossing $s$ twice per unit length.

In the final case, $s$ is crossed near a corner and the bulk of $s$ is inside the convex hull. However, then the two neighbors on either side of the corner must be covered by the sixth case. We charge the triangulation length of $s$ to those neighbors, which have side length at least $l/2$. Each square can be charged at most once by a larger square, and once by an equal- or smaller-sized square. So the extra charge from

this case is at most $1\frac{1}{2}$ times that of the seventh case, or at most 34.5 per unit edge length.

All cases together contribute a charge of at most 316 per unit edge length to the edges of the MWT, so the total edge length of the quadtree triangulation is at most 316 times that of the MWT. □

### 2.4. *Approximation to MWST*

We have shown that the quadtree triangulation approximates the MWT. However, our desired result is that it approximates the MWST.

**Theorem 2.** *The total edge length of the quadtree triangulation is $O(m')$, where $m'$ is the total edge length of the MWST.*

*Proof.* Let $S$ be the initial point set, and let $S'$ be the set of Steiner points added to form the MWST. The MWST is just the MWT of $S \cup S'$. Consider forming a quadtree triangulation in the following way: we first form the quadtree for $S$ as before, and then we add the points in $S'$ and perform further splits as required. Then we add the edges of the convex hull of $S$ and triangulate squares as before.

The extra splits can only increase the length of the triangulation, so this triangulation has a greater total length than the actual quadtree triangulation we would compute for $S$ alone. However, this new triangulation satisfies the assumptions of Lemma 2 for input $S \cup S'$, and so it has length $O(m')$ by that lemma. □

Further, the detailed analysis of Lemma 2 carries over to show that the approximation factor in Theorem 2 is at most 316.

**Corollary 1.** *Quadtree triangulation approximates the MWST of any convex polygon.*

The same result does not carry through for nonconvex polygons. The difficulty is that the MWST of a polygon can have a smaller weight than the MWST of its vertices. It is not difficult to come up with examples in which simply performing the quadtree triangulation of the polygon vertices, then refining the triangulation to include the polygon boundary, can lead to an $\Omega(\log n)$ factor between the resulting triangulation and the MWST. The problem is that the balance condition in the quadtree can cause splitting to spread between regions that are nearby geometrically, but far within the polygon. It seems likely that a more constrained balance condition can be used to develop an algorithm for approximating the MWST of polygons.

## 3. Modified Quadtree Triangulations

The quadtree triangulation above is unsatisfactory as an approximation to the MWST, because it does not run in polynomial time. This difficulty arises because input points may be very close to each other, and so many levels of the quadtree may be constructed before the points are separated. Hence the running time may depend on the geometry of the input point set as well as on the number of points. We must modify our triangulation to avoid small squares.

We also wish to modify the triangulation in another way. Recall that we added edges of the convex hull of the input, and removed all portions of quadtree squares outside that polygon. This leads to many special cases in the details of the triangulation, which are avoided if all input points are interior to the triangulation. In particular, we do not wish to add more cases to the already complicated quadtree algorithms of Bern *et al.* [1] for nonobtuse triangulation and for triangulation without small angles. For this reason, we now describe a version of the quadtree triangulation in which we triangulate both inside and outside the convex hull of the input. The algorithm itself is perhaps even simpler than the previous one, but its analysis is somewhat more complicated, since the exterior of the convex hull can add a total triangulation length proportional to that of the interior but it is not so easy to charge this length against edges of the MWT.

The first step is placing the root square. We rotate the coordinate system so the horizontal axis is aligned with the longest segment connecting two input points. Let $x_1$ and $x_2$ be the minimal and maximal extent of the points in the horizontal direction, and let $y_1$ and $y_2$ be the extent of the points in the vertical direction. We use a root square having a side length equal to the length of that long diagonal, and with a bottom side on the line $y = y_1$. The corners of the square are the four points $(x_1, y_1)$, $(x_2, y_1)$, $(x_1, (x_2 - x_1 + y_1))$, and $(x_2, (x_2 - x_1 + y_1))$. The placement of the root square is illustrated in Fig. 4(a); the long diagonal runs horizontally (because of the coordinate system rotation) and is indicated by a dashed line.

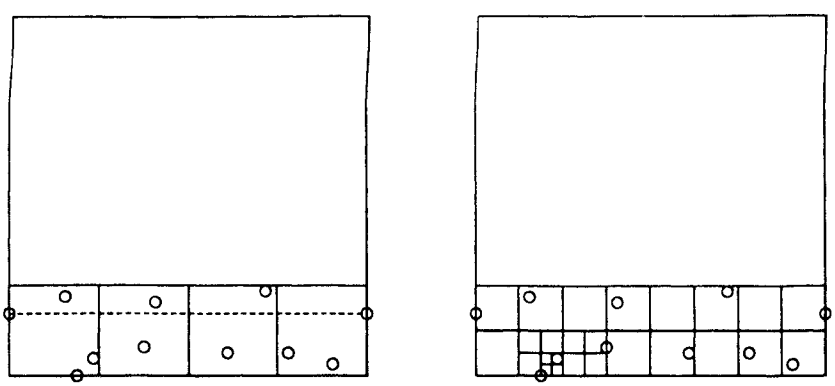



**Fig. 4.** Quadtree triangulation: (a) placement and initial subdivision of the root square, showing the long diagonal; (b) subdivision until points are separated and squares are balanced.

Next, we do some initial subdivision to reduce the area of the quadtree that we must actually triangulate. This step is essential in reducing the approximation ratio from $O(\log n)$ to $O(1)$. Let $k$ be the largest integer for which $(x_2 - x_1)/2^k > (y_2 - y_1)$. Then we subdivide the quadtree $k$ levels deep, so that the leaf squares have size $(x_2 - x_1)/2^k$. By the choice of $k$, all input points will be contained in squares in the bottom row of the quadtree, so we only need triangulate this bottom row. The rest of the quadtree can be ignored for the remainder of the algorithm. The initial subdivision is also illustrated in Fig. 4(a).

We further subdivide squares of the quadtree, maintaining the balance condition. We subdivide any square containing two input points in its interior or on its boundary. Whenever two neighboring squares violate the balance condition, we must also split the larger of the two squares, regardless of whether it contains an input point. At the end of this process, each input point is alone in its square. However, this phase of the algorithm is inherently nonpolynomial, because two points close together may require a number of subdivisions unrelated to the total number of points. The result of this subdivision process is depicted in Fig. 4(b); in the figure only one square has been split because of the balance condition, but in practice this situation may arise more frequently.

Finally, we triangulate individual squares as in the previous algorithm. We do this for all squares, not just those interior to the convex hull of the input.

**Lemma 3.** *The quadtree triangulation described in this section approximates the edge length of the MWST.*

*Proof.* As in Lemma 2 and Theorem 2, we need only prove that it approximates the minimum weight non-Steiner triangulation; the lemma will then follow from the fact that adding Steiner points cannot decrease the total quadtree triangulation length.

Rather than charging empty quadtree squares to MWT edges, we charge all such squares. When four squares of side length $l$ are created by subdividing their parent, a total edge length of $4l$ is added to the quadtree, so each square except for the initial root squares carries a charge of $l$. The root squares can be charged against the boundary of the convex hull of the input. The charge per square is then multiplied by some $O(1)$ factor to account for the edge length added by the diagonals used to form a triangulation.

We reallocate the charge so that the only charged squares are those containing at least one input point. Squares for which a parent contains a point are charged to that parent. Remaining squares must be formed by the balance condition, and for each such square one neighbor of the parent must contain an input point. We charge that neighbor. Each square is charged for the weights of at most 19 smaller squares, so the total charge is proportional to the sum of the charged squares' perimeters.

We next reallocate the charge again so that the only charged squares are subdivided and have between one and three children containing a point. As in Lemma 2 we can deal with unsubdivided squares containing points by further subdividing them until all such squares are very small, and cannot contribute

significantly to the total edge length of the triangulation. If a square is subdivided, but has points in all four children, we divide its charge equally among the children. Any square can get half again its charge from its parent, a quarter from its grandparent, and so on, so the charge per square at most doubles in this stage.

We now divide the problem into cases. The first case consists of squares entirely contained within the convex hull of the input points. Consider one such square $s$, with side length $l$. By assumption $s$ contains an empty child $s'$. Consider the MWT triangle containing this child's centerpoint. Since the child is empty, at least two of the sides of the triangle must have length at least $l/\sqrt{8}$. If a triangle vertex is within distance $l/2$ of $s$, charge the perimeter of $s$ to the longest adjacent triangle edge. Otherwise, some edge of the triangle partitions $s$ into two regions, one containing the centerpoint of the empty child and the other containing the input points in $s$. As in Lemma 2 we can step from triangle to triangle until we find a long MWT edge with a nearby endpoint, on which we can dump the charge carried by $s$. Each MWT edge is charged by $O(1)$ squares of each size, so the total charge is proportional to its length.

The second case consists of squares crossed by the convex hull boundary, but for which a same-size neighboring square is entirely contained in the convex hull. If the neighboring square contains a point, we charge the boundary square to the neighboring square; each interior square is so charged at most eight times. Otherwise, the neighboring square is empty. Again, we have a situation in which a square in the convex hull is empty but has a nearby point. Again, either the MWT triangle containing the square center or some other triangle found by following a path of triangles provides a long edge with a nearby endpoint to charge. As in the first case, each MWT edge is charged proportionally to its length.

In the final case we must account for squares crossing the convex hull boundary, for which all neighbors also cross the convex hull boundary. Because of the initial choice of quadtree position, there are $O(1)$ such squares of each possible size in which the convex hull boundary passes through the top or bottom of the square. For the remaining squares, the boundary in the square and all of its neighbors passes through the left and right sides; therefore the squares of a given size have five similar neighbors, either up or down and to both the left and right (Fig. 5(a)).

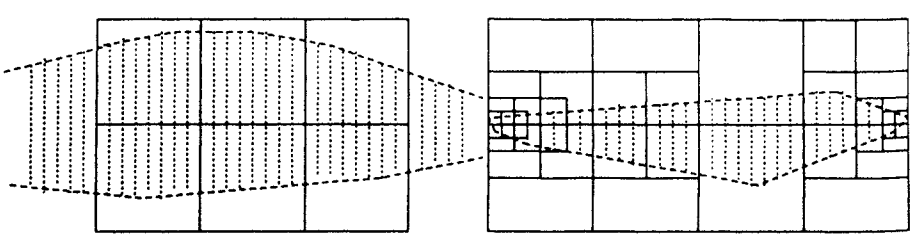



**Fig. 5.** Squares with no neighbor interior to the convex hull: (a) five neighbors, with the convex hull boundary passing horizontally through each; (b) squares of each size form rows near the left and right sides of quadtree.

The three remaining possible neighbors are entirely outside the convex hull. Thus the squares of a given size in this case form double rows, above and below the long diagonal of the input points (Fig. 5(b)). If the initial squares have side length $s$, and the length of the long diagonal is $r$, then because $s$ was chosen as small as possible, the squares of side length $s2^{-k}$ can only occur within distance $r2^{1-k}$ of the left and right sides of the quadtree. The total perimeters of all boxes in this final case is $O(r)$, which is proportional to the convex hull perimeter. Since each convex hull edge is in the MWT, this third case can also be charged to the length of the MWT. □

### 3.1. *Truncated Quadtree Algorithm*

We now address the difficulty noted at the start of this section, that our basic quadtree triangulation algorithm does not run in polynomial time. If, however, it takes more time than $O(n \log n)$, two points must be close together, and the edges between them will contribute little to the total weight of the MWST, or to the weight of the quadtree triangulation. To make this precise we prove the following lemma.

**Lemma 4.** *If we consider a quadtree square of side length $s$, containing $k$ input points, and replace its contents with any triangulation of the points together with the corners of the square and some of its side midpoints, the difference in weights between the portion of the original triangulation contained in the square, and the new triangulation in the square, is $O(ks)$.*

*Proof.* This follows from the fact that the total length of the edges in the replacement triangulation must be $O(ks)$. □

Therefore, from the lemma, if the root square length is $r$, if we modify the quadtree triangulation by not subdividing any square smaller than size $r/n$, we will achieve a total increase in length of $O(r)$ in the unsubdivided squares, together with some possible decrease due to fewer balance subdivisions elsewhere in the quadtree. However, any triangulation must have total length $O(r)$, because that is proportional to the convex hull perimeter of the input points.

**Theorem 3.** *The truncated quadtree triangulation described above can be constructed in time $O(n \log n)$, has $O(n \log n)$ Steiner points, and approximates the MWST.*

*Proof.* If we stop subdividing when the squares reach size $r/n$, the quadtree will have at most $O(\log n)$ levels. Since the squares at each level contain points, are neighbors of squares containing points, or are children of such neighbors, the total number of squares at each level can be shown to be $16n$. Therefore the total size is $O(n \log n)$. Small squares containing several points can be triangulated using a plane-sweep algorithm, in total time $O(n \log n)$. The same argument as for the original quadtree triangulation establishes the time bound for the rest of the construction. □

### 3.2. *No Sharp Angles*

We next examine the triangulation from [1], in which all angles are bounded between 36° and 80°. This triangulation is based on quadtrees as before, with a few notable differences.

- The balance condition is stronger: diagonal neighbors as well as orthogonal neighbors are required to be within a factor of two in size from each other.
- Input points are *well separated* from each other: there must be a certain number of empty squares between them.
- Input points are well separated from the boundary of the quadtree; the quadtree root is chosen so that if the root is divided into 16 squares, all points are contained in the central four squares.
- Empty squares are triangulated by replacing them with one of a certain set of *tiles*; the choice of tile is made by examining the relative sizes of the four orthogonal neighbors. Tiles achieving the angle bounds above are shown in Fig. 6(a); an example triangulation is shown in Fig. 6(b).
- The squares of the quadtree are rearranged near input points so that each point is positioned close to the center of its square; then the square can be triangulated by an appropriate tile, using the input point as one of the vertices.

The strengthened balance condition can be handled by charging empty squares to farther away squares containing points; this changes our analysis by at most a constant factor. Similarly, the triangulation by tiles and the rearrangement near input points multiplies the length by at most a constant factor.

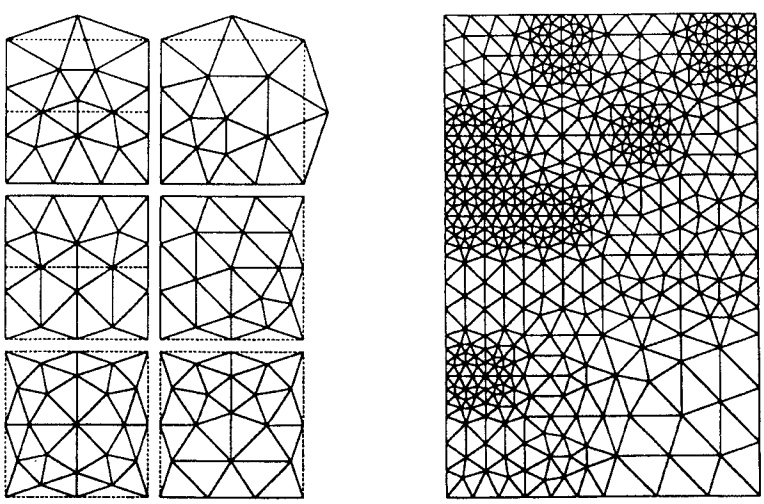



**Fig. 6.** Angle-bounded triangulation: (a) tiles for triangulating empty squares; (b) example triangulation.

The only difference that causes us any difficulty is well separation of points from each other and from the boundaries. Well separation of points from each other is achieved by subdividing squares containing single points, if there are points in other nearby squares. However, our analysis was carried through under the assumption that squares containing points are further subdivided to make their size negligible, so we have already taken into account this well-separation restriction.

Finally, we must choose a root square or collection of root squares, satisfying the separation of input points from root boundaries needed by the algorithm. In the quadtree triangulations above, the root squares contain points exactly on their boundaries. Also, the analysis in [1] of the number of Steiner points depends on the root square having few points, which would seem to conflict with our goal of fitting the root squares to the convex hull of the point set.

Recall that in our original algorithm we chose a sequence of squares, parallel to the long diagonal of the points, with size within a factor of two of the height of the points measured perpendicular to the long diagonal. This choice was used in our analysis, because it limits the total number of squares that touch the convex hull boundary and for which all same-size neighbors also touch the boundary.

If the square size is too small, there will be many squares, even when only a few Steiner points are needed to achieve small angles in the triangulation. We expand the square size, so that there are at most $n$ initial squares; for the same reasons as in the truncated quadtree triangulation, this will not hurt our approximation to the MWST. We then expand the square size slightly, so that the convex hull of the input points is well separated from the outside of the initial squares. Again, this will not hurt our approximation.

The result of this is an initial set of $O(n)$ squares having the properties needed for the bounded angle triangulation. The analysis in [1] together with the appropriately modified proof of Theorem 2 give us the following result.

**Theorem 4.** *For any $\varepsilon$, let $k(\varepsilon)$ be the minimum number of Steiner points required to triangulate a point set with no angle smaller than $\varepsilon$. Then we can find in time $O(n \log n + k)$ a Steiner triangulation with a total length $O(1)$ times the MWST length, in which all angles are bounded between 36° and 80°, and for which the number of Steiner points is $O(k + n)$.*

*Proof.* The construction of the initial set of root squares can be done as before in $O(n \log n)$ time; the analysis of [1] then applies to give the stated bounds on total time, number of Steiner points, and triangle angles. We charge the edge length to three different cases. First, we consider the squares that are larger than the initial root squares of the simpler quadtree triangulation of Theorem 2. There are $O(n)$ squares of each size, and the total length is therefore proportional to the convex hull perimeter of the point set. Second, we consider the length caused by squares splitting because they contain more than one point, or by rebalancing after such splits. However, these squares would exist in either type of triangulation, and the proof of Theorem 2 is not sensitive to the exact alignment of the quadtree box positions, so as in that theorem the total length of these squares is $O(1)$ times

the MWST length. Finally, we consider splits caused by the requirement for well separation. However, each such split is of a square containing a single point, and even if we performed these splits *ad infinitum* instead of stopping after the points were well separated, the total length added would merely sum in a geometric series to an amount proportional to the size of the square containing the point before such splitting. So in this case also the length is $O(1)$ times that of the MWST. □

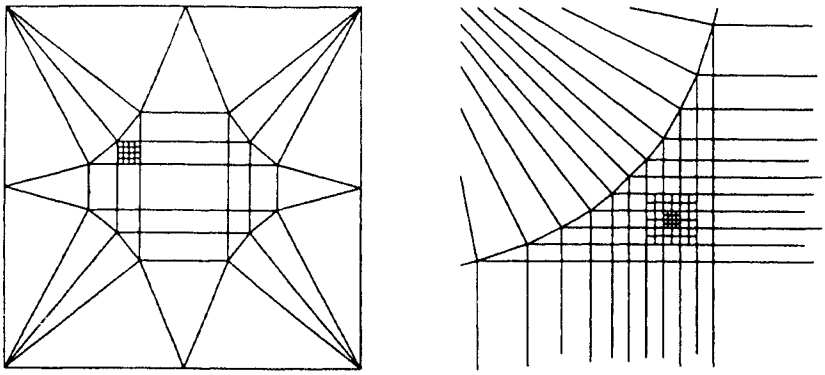
The area covered by the triangulation may be much larger than the convex hull of the input points; this is because of the expansion of the initial squares to reduce the number of initial squares to $O(n)$. We can forgo this expansion, at the expense of increasing the number of Steiner points. Without initial expansion, the number of Steiner points is $O(k' + n)$, where $k'(\varepsilon)$ is the minimum number of Steiner points required to triangulate the points with no angle smaller than $\varepsilon$, and with total area proportional to the convex hull of the input set.

### 3.3. *No Obtuse Angles*

Bern *et al.* [1] describe another triangulation, in which $O(n)$ Steiner points are added and all angles are acute (strictly less than 90°). Such a triangulation must be a Delaunay triangulation of its vertices. We now show how to modify this triangulation to approximate the MWST; again, the modification consists of judicious selection of the initial set of root squares.

The nonobtuse triangulation of [1] uses the insight that the only nonlinear behavior of the quadtree triangulation occurs when a square containing a point set is repeatedly subdivided, without causing the point set to be subdivided also. This can happen because the points are all in the same child square, or because points are in different children but are not well separated. If this happens for the same point set at more than some constant number of levels, we can identify a closely grouped *cluster* of points, which are causing the difficulty, and which are well separated from any other points in the input set. We then *shortcut* the nonlinear behavior of the quadtree by recursively triangulating the cluster, and connecting this recursively constructed quadtree to the outer quadtree.

The connection is formed by embedding the cluster in a small mesh of squares; this prevents subdivision caused by the balance condition from reaching outside the mesh. By some local rearrangements of the outer quadtree squares, we can cause the cluster to be placed near the center of the square containing it. We then *mirror* the cluster, placing copies of it in three locations forming a rectangle within the outer square (Fig. 7(a)). The mirror images contain copies of the mesh of squares but not of the actual input points. We can then connect the recursive quadtree and its mirror images to the outer squares, with a collection of acute and right triangles (Fig. 7(b)). By slightly warping the rectangle of mirror images, all right triangles can be made acute. Squares not containing input points or clusters are triangulated with tiles as in the angle-bounded triangulation. A new

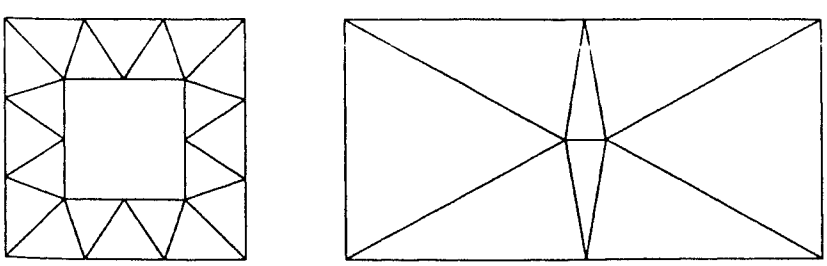


(a) (b)

**Fig. 7.** Attaching the cluster to the outer quadtree: (a) mirroring of the cluster and attachment to the outer square; (b) detail of the attachment.

tile is used to connect the rest of the triangulation with the squares containing clusters (Fig. 8(a)). The connection between clusters and their outer squares uses a large but constant number of edges, since each cluster is formed by a quadtree in which a constant number of subdivisions appear on the boundary of the initial root square. The length of these edges can therefore be charged to the length of the outer square.

As in the angle-bounded triangulation, the strengthened balance condition and the requirement of well-separated points can be dealt with at a constant factor cost in the approximation constant. We could choose initial squares as in that triangulation, but in fact we can do better in this problem. Recall that, in our basic quadtree triangulation algorithm, we choose a set of initial squares aligned with the long diagonal of the input points and with side length proportional to



(a) (b)

**Fig. 8.** Extra tiles for nonobtuse triangulation: (a) connection to the cluster; (b) acute triangulation of a rectangle.

the maximum distance of any point from that long diagonal. Denote the number of initial squares by $m$. If $m < n$, the number of Steiner points formed by the corners of the initial squares will not interfere with our total bound. Otherwise, at most $n$ of the squares can actually contain a point. For each such square, we also keep two empty squares on either side; the outer empty square will remain undivided by balance condition requirements. This gives us a collection of $5n$ initial squares. The remaining $m - 5n$ squares are merged into a set of at most $n$ rectangles, simply by removing the boundary edges separating them. Each rectangle will touch on either side an unsubdivided square, which (because of the replacement by tiles) will have two equally spaced points on its boundary. Thus the rectangle can be split into three narrow rectangles, each of which can be triangulated with acute triangles by adding Steiner points at the midpoints of each long rectangle side and two Steiner points in the interior of each rectangle (Fig. 8(b)).

**Theorem 5.** *We can find in time $O(n \log n)$ a Steiner triangulation in which all angles are less than 90°, with a total length $O(1)$ times the MWST length, covering an area $O(1)$ times the convex hull of the input points, and for which the number of Steiner points is $O(n)$.*

*Proof.* The proof that angles are acute, that the number of Steiner points is $O(n)$, and that the time is $O(n \log n)$, are all the same as in [1]. The remaining difficulty in proving approximation to the MWST is that, by recursively triangulating clusters, we lose the property that adding Steiner points can only further subdivide the quadtree squares. The problem is that the Steiner points can cause the cluster not to be separated from the main quadtree, or cause the cluster to have a larger convex hull, in either case leading to squares with a different alignment.

Instead, we bound the total length indirectly, by comparing it with the previously described quadtree triangulation of Theorem 4. Note that in all quadtree triangulations we have described, any unsubdivided square not containing an input point will have a side length proportional to the distance from its center to the nearest input point. This distance is clearly a lower bound on the side length, and it is an upper bound because no operation used to construct our quadtree triangulations subdivides squares far away from any point. In both the triangulation considered here and that of Theorem 4, any unsubdivided square containing an input point will have a side length proportional to the distance from that point to the nearest square. This follows simply because we subdivide squares until the points are well separated, and no further. So we can bound the edge length of the nonobtuse triangulation considered here, by charging each unsubdivided quadtree square to the square containing its centerpoint in the triangulation of Theorem 4. Each square in that triangulation will have charged to it $O(1)$ squares of similar sizes in the nonobtuse triangulation, and therefore the nonobtuse triangulation also approximates the MWST length. □

## 4. Properties of the MWST

### 4.1. *MWST and MWT*

Clearly, the MWST must have a total edge length of at most that of the MWT. It is also not hard to show that the weight of the MWT (or any other non-Steiner triangulation) is $O(n)$ times the perimeter of the convex hull, and therefore $O(n)$ times the MWST weight. As we now show, this latter factor can be tight, even for the MWT.

**Theorem 6.** *For any $n$, point sets exist for which the MWT has weight $\Omega(n)$ times the MWST weight.*

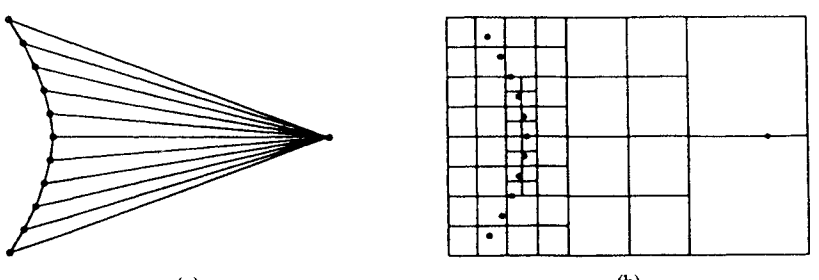
*Proof.* Start with an isosceles triangle, with two long side lengths equal to some value $y$ and one short side length equal to $x$. Let the short side be replaced by a concave chain of $n-1$ vertices, giving (with the remaining vertex at the apex of the triangle) a total of $n$. Then the only empty triangles touching the apex are those formed by pairs of adjacent vertices in the concave chain; therefore all such triangles must be present in the MWT, and the total MWT length is $\Omega(yn)$ (Fig. 9(a)).

However, the quadtree triangulation will contain subdivided squares of size $s$ only within distance $O(s)$ of the concave chain. There can be at most $O(x/s)$ such squares, for a total length of $O(x)$ at each level of subdivision. As usual we need only count squares of side length $\Omega(y/n)$, for which there are $O(\log n)$ possible sizes. The total length of the quadtree triangulation, and hence the MWST, is $O(y + x \log n)$ (Fig. 9(b)).

If we let $x = y/\log n$ we achieve a ratio of $\Omega(n)$ between the MWT and MWST lengths as desired. □

### 4.2. *MWST and MST*

The algorithm of Clarkson [2] shows that there exist Steiner triangulations with a total weight within an $O(\log n)$ factor of the MST weight. A similar result holds for our triangulations (see Lemma 10 below). Therefore the MWST weight is within a logarithmic factor of the MST weight. We now show that this is tight.



(a) (b)

**Fig. 9.** Point set illustrating the ratio of MWT to MWST lengths: (a) MWT length $\Omega(yn)$; (b) quadtree length $O(y + x \log n)$.

**Theorem 7.** *For any $n$, point sets exist for which the MWST has a weight $\Omega(\log n)$ times the MST weight.*

*Proof.* Let the points be equally spaced around the unit circle. Then the MST weight is approximately $2\pi(1 - 1/n) = O(1)$. The quadtree triangulation will produce squares of all sizes down to the level at which the points are separated from each other; this occurs when the side length nears $1/n$, at a level $\Omega(\log n)$. For each side length $s$, there will be approximately $1/s$ unsubdivided squares of that size, created because of the balance rule, in two chains running around the inside and outside of the unit circle. The total length of these squares is $\Omega(1)$. Therefore the total length of all unsubdivided squares, and the total length of the quadtree triangulation, is $\Omega(\log n)$. However, since the quadtree triangulation approximates the MWST, the MWST length must also be $\Omega(\log n)$. □

### *4.3. Convex Polygons*

We now consider the MWST for a convex polygon. We have already noted in Corollary 1 that such an MWST can be approximated using a quadtree-based algorithm. It is at least plausible that the MWST of a convex polygon is approximated by its MWT (without Steiner points); first we show that this is not true.

**Theorem 8.** *For any $n$, convex polygons exist for which the MWST has weight $\Omega(\log n)$ times the MWT weight.*

*Proof.* Start with $n$ evenly spaced points on the unit interval, and raise them slightly so that they form a single convex chain. The unit interval itself is then an edge in the convex hull. Then by adding the original unraised points as Steiner points on the unit interval, we can produce a triangulation with total length approximately 3. If we have no Steiner points, the MWT can be found as a triangle based on that long edge, together with two MWTs of smaller polygons formed by taking shorter convex chains. The total length can be described by a recurrence

$$L(m) = O(m/n) + \min_{a+b=m+1} L(a) + L(b)$$

which solves to $\Theta(\log n)$. □

So we need Steiner points even to approximate the minimum length. However, in the example above, all the Steiner points fell on the polygon's boundary. We conjecture that, in fact, the MWST of any convex polygon needs only to use Steiner points on the polygon's boundary. This is false for nonconvex polygons, as can be seen for the polygon in Fig. 9(a). The truth of our conjecture would imply a polynomial-time algorithm for computing the MWST in this case. As a partial result in this direction, we show that boundary Steiner points can be used to approximate the MWST.

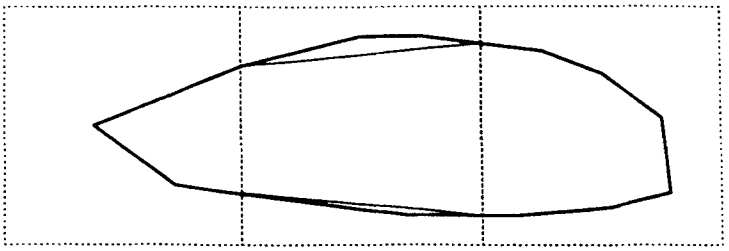


**Fig. 10.** Initial root boxes and diagonals for triangulation with no interior Steiner points.

**Theorem 9.** *The MWT using Steiner points only on the boundary of a convex polygon has weight $O(1)$ times the MWST of the polygon.*

*Proof.* We derive such a triangulation from our quadtree-based triangulation algorithms. We start, as in the quadtree algorithms, by finding a horizontal row of squares containing the polygon. Each square cuts the polygon by two vertical lines; these cuts can be represented as diagonals between Steiner points on the boundary of the polygon.

Each initial square will contain portions of both the upper and lower boundary of the polygon. We next add diagonals connecting the points where the initial quadtree square edges cross the input boundary. Figure 10 depicts these initial squares and diagonals. We do not add edges crossing these diagonals, so we can treat the upper and lower portions of the boundary independently of each other, and work as if each quadtree square contains a single connected component of the polygon boundary. The total length of the initial squares and diagonals is proportional to the perimeter of the polygon.

Finally, we follow the quadtree algorithm, splitting squares until points are alone in their squares. As we split, each square will contain a connected portion of the input polygon boundary, with the endpoints of the boundary segments connected by a diagonal. These regions between the diagonals and the boundary segments will be the only untriangulated areas in the polygon. When we split a square or rectangle in two, the lines along which we split may cut the polygon boundary in one or more places. We then add new Steiner points at the cut points, and connect them by diagonals to the endpoints of the boundary segment. This partitions the polygon into a region between the new and old diagonals, and a number of subproblems in the region between the new diagonals and the polygon boundary. We triangulate the region between the new and old diagonals by adding $O(1)$ extra diagonals, and we continue recursively in the remaining regions, each of which will be contained in a single quadtree square. Some squares may again contain two separate components of polygon boundary, but again they will be separated by diagonals and can be treated independently of each other. Every time we split a square, the total length added to the triangulation is proportional to the length added to the quadtree. Therefore the total length of the triangulation approximates the MWST length. □

## 5. Extensions to Higher Dimensions

An obvious question arises from these results: do they carry over into higher dimensions? Can quadtree triangulations approximate the MWT of higher-dimensional point sets?

To answer this question, we first have to determine what we mean by the MWT. Given a triangulation, we can measure its 0-faces (points), 1-faces (edges), etc., up through $d$-faces. The number of 0-faces must be at least $n$, and any non-Steiner triangulation will have that many. The $d$-faces must have a total measure of at least that of the convex hull, and again any non-Steiner triangulation achieves this. That leaves $d - 1$ possible functions to minimize. In two dimensions the only interesting minimization problem is that of the total edge length. However, even in three dimensions, both edge length and triangle area can be examined.

Note that, in two dimensions, our triangulations not only optimize edge length, but at least one of them (the nonobtuse triangulation) also uses $O(n)$ Steiner points and fits within an area $O(1)$ times the area of the original point set. Thus it simultaneously optimizes all three possible criteria. However, we saw that this simultaneous optimization must be modified if we desire further properties, such as no small angles. Then we must count the optimal number of Steiner points over triangulations also having these properties. Nevertheless, it is reasonable to hope that similar simultaneous approximation results hold in higher dimensions.

Let us examine the three-dimensional case in more detail. First we must determine what to use as the root box of our octtree (three-dimensional quadtree). If we wish to approximate the minimum triangle area, we must closely fit our initial set of cubes to the convex hull of the point set; otherwise the surface area of our initial set will be too large. On the other hand, if we start with an initial set of cubes that are too small, the total edge length would be too large. It seems that already in the initial placement we need some sort of hierarchical decomposition.

Next we examine our proof of optimality. The main step was amortizing cubes with an empty child against MWT edges with a nearby endpoint. We did this by starting at the triangle containing the child's centerpoint. If the endpoints were not nearby, one face of the triangle separates the centerpoint from the points in the square, and we can use that fact to move from there to a large triangle with a nearby vertex. In three dimensions we can similarly reduce the problem to counting squares with an empty child. The tetrahedron containing the child's centerpoint will have sufficiently long edges, but it may be long and narrow, having a surface area that is arbitrarily small. Indeed, it may be that no triangle has an area proportional to that of the cube. The triangle's edges will be long, but if the endpoints are not nearby it will not be clear in which direction to move in order to find a nearby triangle with long edges.

Given these difficulties, it is surprising that we can find any higher-dimensional generalization of our results. However, in fact we can prove the following, which shows that quadtree triangulations give a (nonconstant-factor) approximation to the minimum edge length triangulation.

**Theorem 10.** *In any dimension d, the appropriate generalizations of the quadtree and truncated quadtree triangulations, starting from a single root box, use total edge length* $\exp(cd)\log n$ *times the length of the MST for some constant c.*

*Proof.* As before, we can charge the length of boxes not containing points to nearby boxes containing points. Each box is charged $\exp(O(d))$ times its length. Then we need merely sum the edge lengths in boxes containing points. The boxes with edge length $1/n$ times the initial root box take a total length proportional to the root box length, so we need only look at higher levels in the quadtree. There are $O(\log n)$ such levels; we show that the length in each is proportional to the MST length.

Consider a box of side length $s$, containing a point $x$. There are $3^d - 1$ neighboring boxes. There are $O(3^d)$ boxes large enough that the entire point set is contained in these neighbors; the total length of these boxes is $\exp(O(d))$ times the MST length. Otherwise, $x$ is connected by the MST to a point outside the neighboring boxes; therefore there must be an MST path from $x$ to that point, and the length of the intersection of that path with the neighboring boxes must be $O(s)$. We charge that portion of the path with the weight of the box; each portion of the MST is charged $2^d$ times its length for as many as $3^d$ boxes at each of the $O(\log n)$ levels. □

As a corollary, $d$-dimensional quadtree triangulation approximates the minimum edge length Steiner triangulation within a factor of $O(\log n)$. This argument also applies in two dimensions, and proves that the triangulations of [1] (in which the root box was not fitted as closely to the input points as it is in our constructions) also achieve this $O(\log n)$ approximation.

We do not know any similar approximation results for the measures of $k$-faces in the triangulation, $k > 1$.

## 6. Conclusions

We have shown that quadtree triangulations achieve a small total edge length. This complements their previously known ability to restrict the angles used in the triangulation, and is further evidence of their utility in finite-element mesh-generation applications. We also used quadtree approximations to the MWST as a tool in proving other properties of the MWST.

In an earlier conference version of this paper [8], we gave a relatively sloppy analysis of the approximation ratio, and did not explicitly derive a numeric approximation ratio. The analysis in this paper has been tightened by perhaps a factor of 100, and yet the constant factor we end up with (316) is still quite large. We believe the constant factor achieved by quadtrees is much smaller, perhaps on the order of 20, but proving this will require an even more careful analysis. Further improvements may come from tuning the algorithms in various ways, from heuristic procedures such as removing Steiner points that do not contribute to decreased length, and from performing various other local optimizations.

Another important open question is whether it is possible to approximate the MWT. At first glance, quadtrees seem unlikely to help, because they use Steiner points and can achieve much lower lengths than the MWT. Hence it would seem difficult to rearrange a quadtree triangulation into a non-Steiner triangulation that approximates the MWT. Plaisted and Hong [21] conjecture that their algorithm computes an approximation to the MWT, but it seems difficult to determine when their $O(\log n)$ factor lost due to the ring heuristic is a necessary part of the MWT, and when it is an artifact of their construction. Since in the MWST problem the same $O(\log n)$ factors sometimes appear, and are easily explained with our quadtree approximation, perhaps our quadtree-based techniques can be used to solve this problem.

## Acknowledgments

I would like to thank Mike Dillencourt for carefully reading a draft of this paper, and for many helpful discussions. I would also like to thank Marshall Bern and Scott Mitchell for clarifying a number of the details needed for efficient implementation of these algorithms.

## References

1. M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990, pp. 231–241. Some cited results appear only in the full version, *J. Comput. System. Sci.*, to appear.
2. K. Clarkson. Approximation algorithms for planar traveling salesman tours and minimum-length triangulations. *Proc. 2nd ACM–SIAM Symp. on Discrete Algorithms*, 1991, pp. 17–23.
3. E. F. D'Azevedo and R. B. Simpson. On optimal interpolation triangle incidences. *SIAM J. Sci. Statist. Comput.* **10** (1989), 1063–1075.
4. D. Z. Du and F. K. Hwang. An approach to proving lower bounds: solution of Gilbert–Pollack's conjecture on Steiner ratio. *Proc. 31st IEEE Symp. Foundations of Computer Science*, 1990, pp. 76–85.
5. H. Edelsbrunner and T. S. Tan. A quadratic time algorithm for the minmax length triangulation. *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, 1991, pp. 414–423.
6. H. Edelsbrunner, T. S. Tan, and R. Waupotitsch. A polynomial time algorithm for the minmax angle triangulation. *Proc. 6th ACM Symp. on Computational Geometry*, 1990, pp. 44–52.
7. D. Eppstein. The farthest point Delaunay triangulation minimizes angles. *Comput. Geom. Theory Appl.* **1** (1992), 143–148.
8. D. Eppstein. Approximating the minimum weight triangulation. *Proc. 2nd ACM–SIAM Symp. on Discrete Algorithms*, 1992, pp. 48–57.
9. M. R. Garey and D. S. Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA, 1979.
10. P. D. Gilbert. New results in planar triangulations. Report R-850, Coordinated Science Laboratory, University of Illinois (1979).
11. D. G. Kirkpatrick. A note on Delaunay and optimal triangulations. *Inform. Process. Lett.* **10** (1980), 127–128.
12. G. T. Klincsek. Minimal triangulations of polygonal domains. *Ann. Discrete Math.* **9** (1980), 121–123.
13. C. Levcopoulos. An $\Omega(\sqrt{n})$ lower bound for non-optimality of the greedy triangulation. *Inform. Process. Lett.* **25** (1987), 247–251.

14. C. Levcopoulos and A. Lingas. On approximation behavior of the greedy triangulation for convex polygons. *Algorithmica* **2** (1987), 175–193.
15. C. Levcopoulos and A. Lingas. Fast algorithms for greedy triangulation. *Proc. 2nd Scadinavian Workshop on Algorithm Theory*, LNCS, vol. 447, Springer-Verlag, Berlin, 1990, pp. 238–250.
16. A. Lingas. Advances in minimum weight triangulation. Ph.D. thesis, Linköping University (1983).
17. E. L. Lloyd. On triangulations of a set of points in the plane. *Proc. 18th IEEE Symp. on Foundations of Computer Science*, 1977, pp. 228–240.
18. R. Löhner. Some useful data structures for the generation of unstructured grids. *Comm. Appl. Numerical Methods* **4** (1988), 123–135.
19. G. K. Manacher and A. L. Zobrist. Neither the greedy nor the Delaunay triangulation approximates the optimum. *Inform. Process. Lett.* **9** (1979), 31–34.
20. D. M. Mount and A. Saalfeld. Globally-equiangular triangulations of co-circular points in $O(n \log n)$ time. *Proc. 4th ACM Symp. on Computational Geometry* 1988, pp. 143–152.
21. D. A. Plaisted and J. Hong. A heuristic triangulation algorithm. *J. Algorithms* **8** (1987), 405–437.
22. W. C. Rheinboldt and C. K. Mesztenyi, On a data structure for adaptive finite element mesh refinements. *ACM Trans. Math. Software* **6** (1980), 166–187.
23. H. Samet. The quadtree and related hierarchical data structures. *Comput. Surveys* **16** (1984), 188–260.
24. R. Sibson. Locally equiangular triangulations. *Comput. J.* **21** (1978), 243–245.
25. W. D. Smith. Implementing the Plaisted–Hong min-length plane triangulation heuristic. Manuscript cited by [2] (1989).
26. M. A. Yerry and M. S. Shephard. A modified quadtree approach to finite element mesh generation. *IEEE Comput. Graphics and Applications* **3** (1983), 39–46.

*Received March 30, 1992, and in revised form December 1, 1992.*